A Parallel Algorithm for the Single-Source Shortest Path Problem

Kevin Kelley, Tao Schardl

May 12, 2010

Outline

- The Problem
- Dijkstra's Algorithm
- Gabow's Scaling Algorithm
- Optimizing Gabow
- Parallelizing Gabow
- Theoretical Performance of Gabow
- Empirical Performance of Gabow
- Future Work

The Problem

Single Source Shortest Paths (SSSP) Given a graph $G = (V, E)$ with non-negative edge weights and a starting vertex v_0, find the shortest path from v_0 to every $v \in V$.

Some Definitions:

- The *weight* of a path is the sum of the weights of the edges along that path.
- The *length* of a path is the number of edges along the path.
- A "shortest path" from v_0 is the *minimum weight path* from v_0.
- The *distance* from v_0 to v, $v.dist$, is the sum of the weights on the minimum weight path from v_0 to v.

Dijkstra's Algorithm

Use a priority queue keyed on distance.

1. Set $v_0.dist = 0$ and $v.dist = \infty$ for all $v \neq v_0$.
2. Create priority queue Q on all vertices in V.
3. While Q is not empty:
    3.1 $v = \textsc{Extract-Min}(Q)$
    3.2 For all u such that $(v, u) \in E$
        3.2.1 $u.dist = v.dist + w(v, u)$, where $w(v, u)$ is the weight of edge (v, u).
        3.2.2 $\textsc{Decrease-Key}(Q, u, u.dist)$

Running Time: $O(E \cdot T_{\textsc{Decrease-Key}} + V \cdot T_{\textsc{Extract-Min}})$.
Can we parallelize Dijkstra's Algorithm?

- Priority queue is a serial bottleneck.
- Only definitely useful operation is to process the minimum element of priority queue at each step.

Gabow's Scaling Algorithm

Idea: Consider the edge weights one bit at a time.

- The weight of the minimum weight path from v_0 to v using just the most significant bit of the weight is an approximation for the weight of the minimum weight path from v_0 to v.
- Incrementally introduce additional bits of the weight to refine our approximation of the minimum weight paths.
- Once all of the bits of the weights are considered, we're done.

Gabow's Scaling Algorithm

- At each iteration, for some edge (u, v) we define the difference in approximate distances $u.dist - v.dist$ to be the *potential* across (u, v).
- We define the *cost* of an edge to be its refined weight at some iteration plus the potential across it: $l_i(u, v) = w_i(u, v) + u.dist - v.dist$.
- Since the sum of costs along a path telescopes, these costs preserve the minimum weight paths in the graph.
- We guarantee that the cost of an edge is always nonnegative.
- $=>$ We can repeatedly find minimum weight paths on graphs of cost values.

Optimizing Gabow

We can restrict the size of the priority queue used on each step.

- The length of a path with p edges can increase by at most p on each subsequent iteration of Gabow.
- Let $p_{i,\max}$ be the length of the longest minimum weight path after the ith iteration of Gabow.
- The sum of the costs on a minimum weight path during the $i+1$st iteration can be no more than $p_{i,\max}$.
- The ith iteration of Gabow can find the minimum weight paths using a monotone priority queue with only $p_{i-1,\max}$ bins.

Parallelizing Gabow

Can we do it?

- The priority queue must store V items in $p_{i,\max}$ bins.
- $p_{i,\max} \leq V$, but we expect $p_{i,\max} < V$ in many cases.
- $\Rightarrow$ We expect bins to contain multiple items.
- We can process the contents of each bin in parallel.

Parallelizing Gabow

Issues with parallelizing Gabow:

- Parallel threads will try to set the distance for a vertex simultaneously. We want the minimum distance to win.
- Parallel threads will be adding vertices to a priority queue in parallel. We want the priority queue to work properly anyway.
- A vertex may have many neighbors connected with zero-length edges. We need to manage these neighbors efficiently.

Parallelizing Gabow

Race condition for distance value: "Double-setting"

- Let the race be.
- When removing a vertex from its minimum bin in the priority queue, ensure its distance value is correct before proceeding.
- At the point when a vertex is removed from its minimum bin, we know its correct distance.
- $\Rightarrow$ The non-benign race becomes benign.

Parallelizing Gabow

Parallel priority queue:

- Don't use a DECREASE-KEY operation; just INSERT.
- When we encounter a vertex we have evaluated already, skip it.
- Currently, we used a locked data structure for each bin to resolve a race for inserting into the same bin.
- Alternatively, use TLS for each bin to remove contention on writing to the same bin.
- Parallel threads can insert into the queue with no contention.

Parallelizing Gabow

Zero-weight edges:

- Keep two buffers for each bin. Fill the second while processing the first.
- Once the first is done, if the second is non-empty, swap the buffers and repeat.
- If the second buffer gets sufficiently large, spawn off a separate thread to process it.

Theoretical Performance of Gabow

Let $G = (V, E)$ be a simple connected weighted directed graph. Let W be the maximum edge weight in G. Let Δ be the maximum out-degree of a vertex $v \in V$.

- Work: $\Theta(E \lg W)$.
- Span: $O(V \lg W \lg \Delta)$ worst-case.
  - Bits of weight are processed serially in phases. $\Theta(\lg W)$
  - Within each phase, each bin in the priority queue is processed serially.
  - Within a bin, the longest chain of vertices connected by zero-weight edges is processed serially.
  - Edges on minimum weight paths from previous phase may have weight of 0 or 1.
  - In the worst case, every vertex appears in some bin's longest chain of zero-weight edges once.
  - Total length of zero-weight edge chains in all bins is $O(V)$ worst case.
  - Each vertex has Δ neighbors to explore, which requires $O(\lg \Delta)$ span.

Theoretical Performance of Gabow

Suppose we have random edge weights, and let D be the length of the longest minimum weight path in any phase of Gabow.

- Work: $\Theta(E \lg W)$.
- Span: $O(D \lg W \lg \Delta \lg V/D)$
  - Each phase must examine D bins serially.
  - The length of the longest zero-weight edge chain in a bin is $O(\lg V/D)$ with high probability.
  - Total length of zero-weight edge chains in all bins is $O(D \lg V/D)$.
- $\Omega(E/V \lg \Delta)$ parallelism worst-case.
- $\Omega(E/(D \lg \Delta \lg V/D))$ parallelism with random edge weights.

Empirical Performance of Gabow

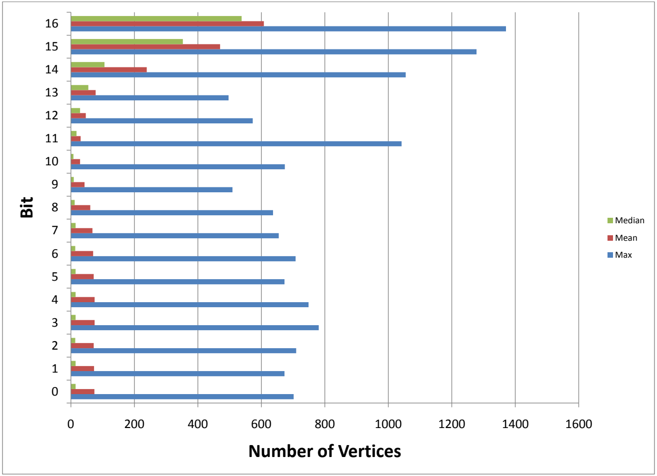
We tested our parallel Gabow implementation on a few input graphs, including the New York and San Francisco Bay road networks.

- First, we collected metrics on the priority queue data structure during Gabow's execution, including Bin size, Queue size, and longest zero-weight edge chain.
- Second, we compared Gabow's serial and parallel performance to a simple Dijkstra implementation.

The data presented here comes from running Gabow on the San Francisco Bay road network. $V = 321270$, $E = 800172$. Parallelism according to Cilkview: 4.76 (2.29 burdened)

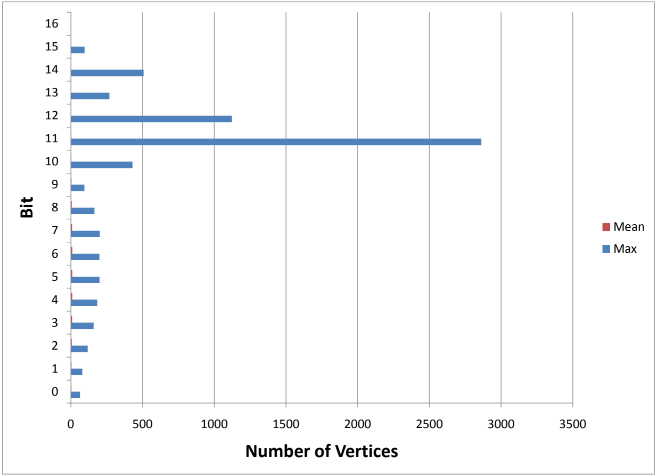
Empirical Performance of Gabow

Number of Evaluated Vertices in Each Bin (San Francisco Bay road network)



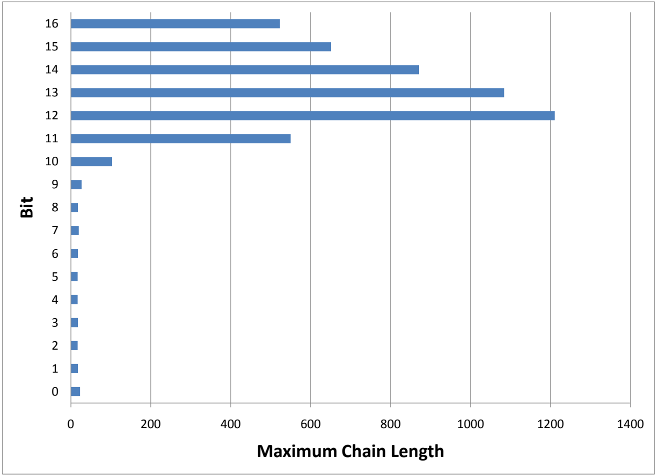
Empirical Performance of Gabow

Number of Ignored Vertices in Each Bin (San Francisco Bay road network)



Empirical Performance of Gabow

Maximum Length of Zero-Weight Edge Chain (San Francisco Bay road network)



Empirical Performance of Gabow

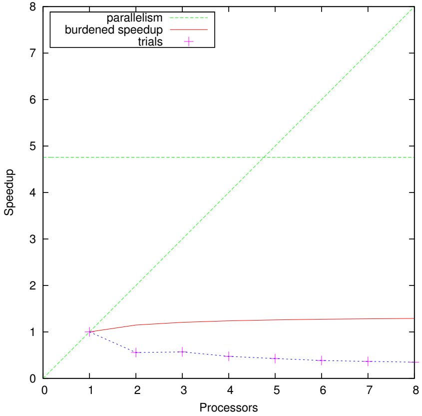
Queue Size (San Francisco Bay road network):

Min	523
Median	1026
Mean	20119
Max	$321270 = V$

Empirical Performance of Gabow

Performance on San Francisco Bay road network:

Dijkstra (ms)	Gabow, 1 proc (ms)
791	5116

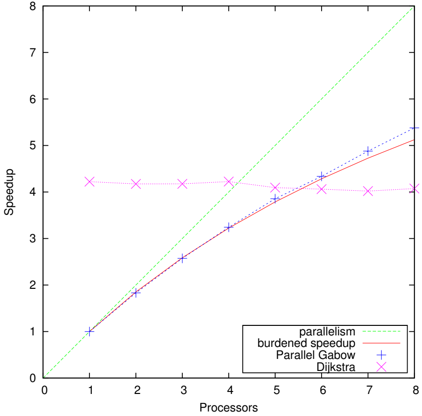


Future Work

- Remove lingering unnecessary serial code in parallel Gabow implementation.
- Use TLS for each bin in the priority queue, rather than a locked vector, to remove lingering contention.
- Investigate memory bandwidth issues.
- Experiment with alternative graph layouts.

Empirical Performance of Gabow as of 05-10-2010

Performance on random graph, $V = 1.5\mathrm{M}, E = 4\mathrm{M}$



Empirical Performance of Gabow as of 05-10-2010

Performance on road network for northeastern U.S.